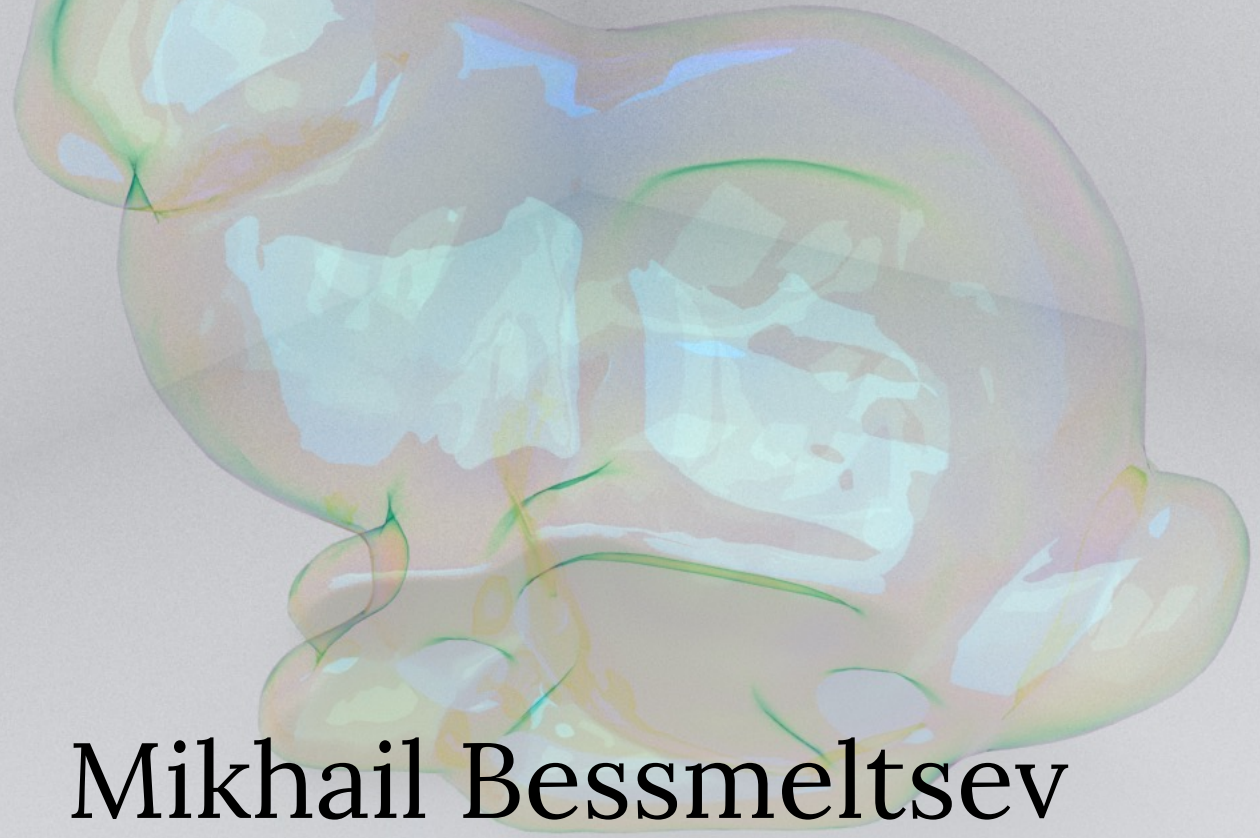
http://tinyurl.com/ift3355

IFT 3355: INFOGRAPHIE

SHADERS, OPENGL, JS

Livre de référence: G:Appendix A*

(pas la même version d'OpenGL)



Mikhail Bessmeltsev

DEVOIR 1

- sera disponible sur StudiuM aujourd'hui
- Date limite: 27 septembre à 23:59:59
- Jours de tolérance: 3 par trimestre
  - Utilisez-les judicieusement
  - Le week-end ne compte pas
    - E.g. la date limite est vendredi → lundi 23:59:59 est **1** jour

- La configuration de l'environnement prend du temps

- **Amusez-vous bien!**

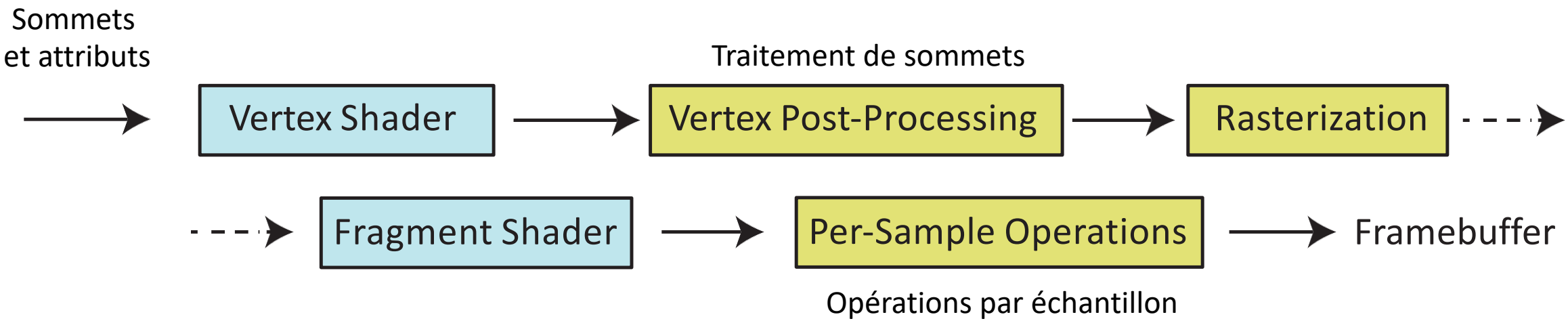
ENVIRONNEMENT

- Écrivez du code dans n'importe quel éditeur
  - Notepad++ (windows)
  - Sublime text (toute plateforme)
  - vim (linux)
  - OU utilisez JetBrains Webstorm (gratuit pour les étudiant.e.s)

- StudiuM

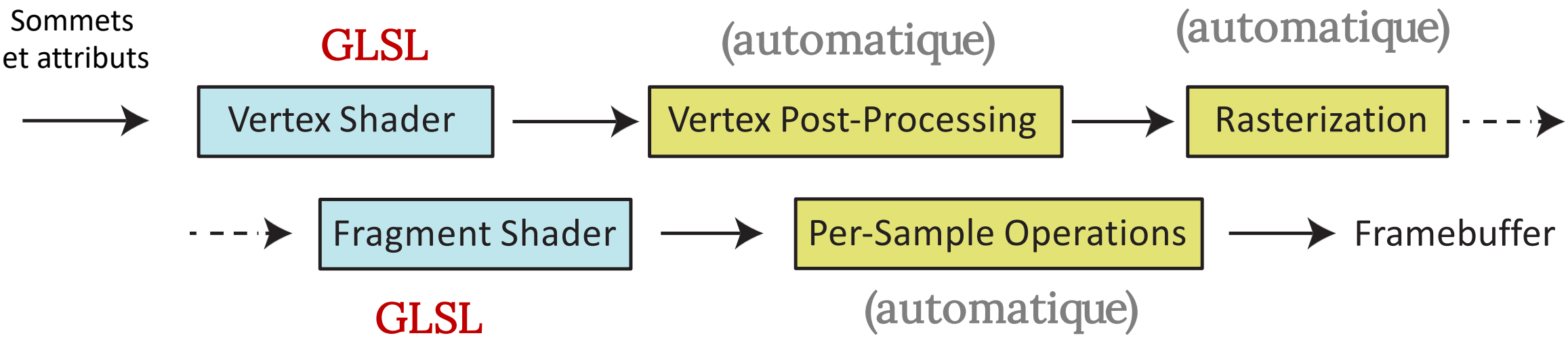
- Les démos

OPENGL PIPELINE DE RENDU



OPENGL PIPELINE DE RENDU

Javascript
+ Three.JS



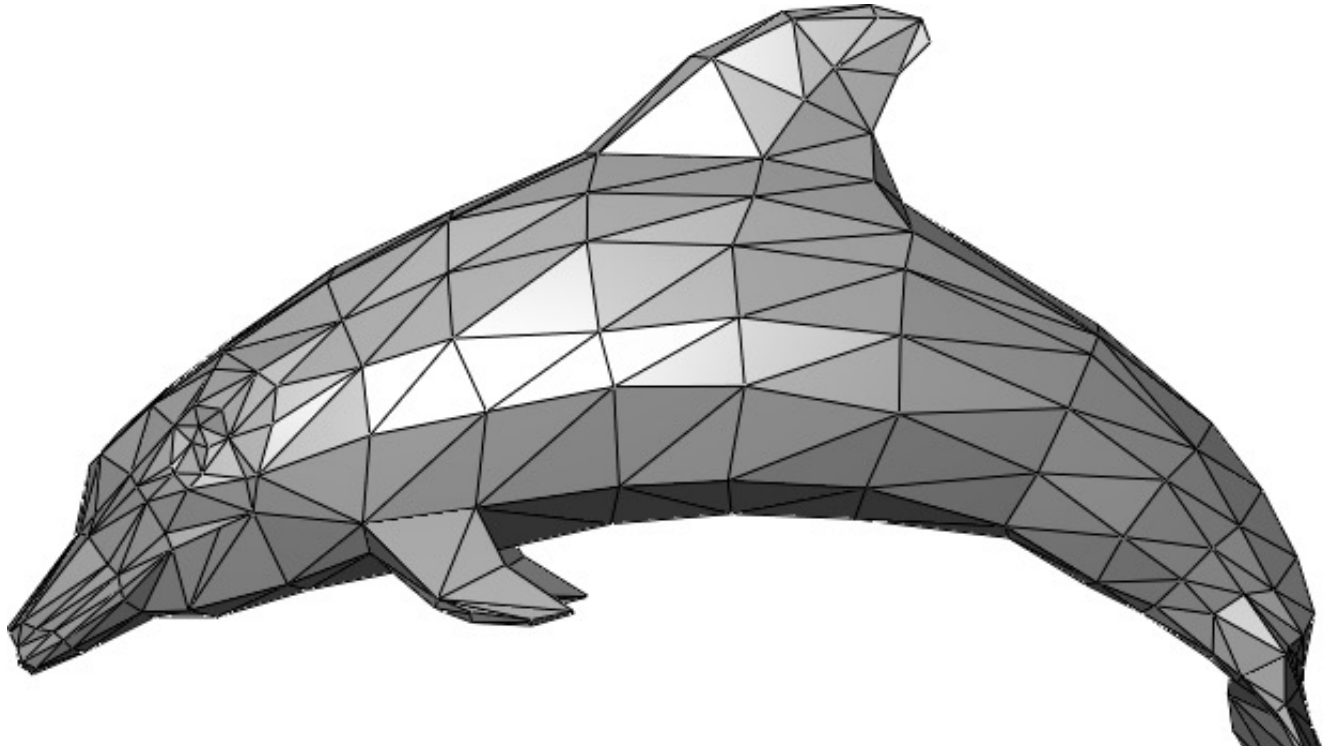
THREE.JS

- Une bibliothèque de haut niveau pour Javascript
- Utilise WebGL pour le rendu

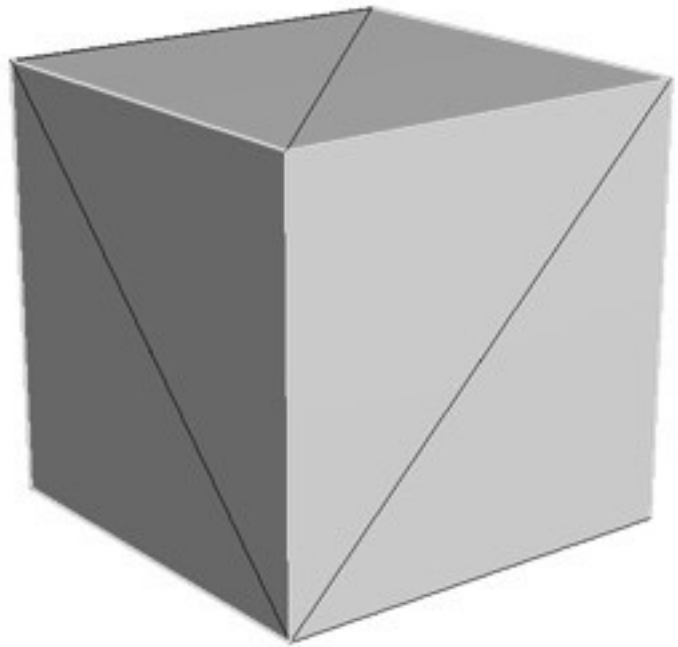
- Contient les objets: **Scene**, **Mesh, Camera**
- **Scene** est hiérarchique
- **Mesh** a la géométrie et les propriétés matérielles
- **Camera** est utilisée pour le rendu

LA GÉOMÉTRIE

- Les maillages de triangles
  - La liste de sommets
  - Les triangles sont définis comme {vertex_index1, vertex_index2, vertex_index3}



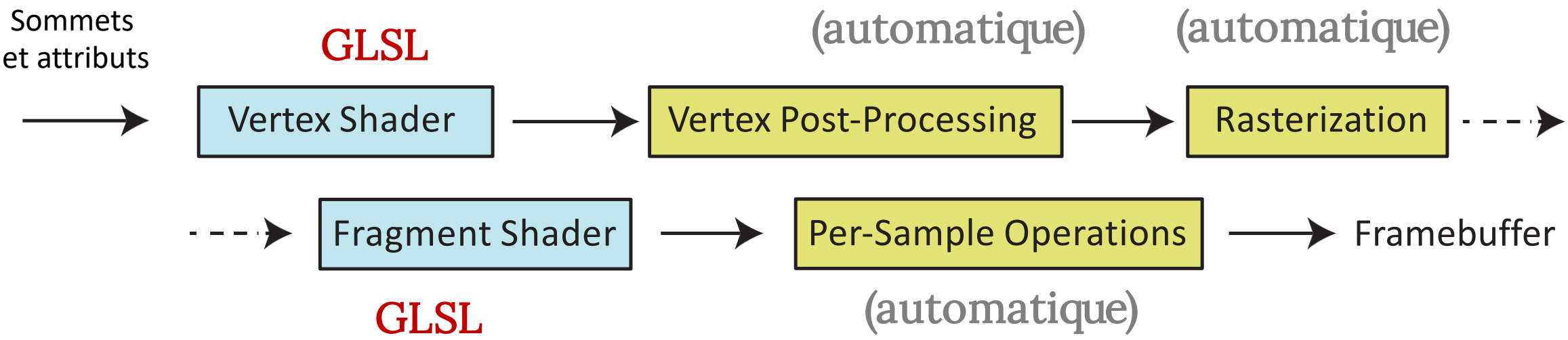
```
var verticesOfCube = [
    -1,-1,-1,    1,-1,-1,    1, 1,-1,    -1, 1,-1,
    -1,-1, 1,    1,-1, 1,    1, 1, 1,    -1, 1, 1,
];
var indicesOfFaces = [
    2,1,0,    0,3,2,
    0,4,7,    7,3,0,
    0,1,5,    5,4,0,
    1,2,6,    6,5,1,
    2,3,7,    7,6,2,
    4,5,6,    6,7,4
];
var geometry = new THREE.PolyhedronGeometry(
verticesOfCube, indicesOfFaces, 6, 2 );
```



GÉOMÉTRIE (JAVASCRIPT/THRE.JS)

OPENGL PIPELINE DE RENDU

Javascript
+ Three.JS



GLSL

- Un langage d'OpenGL pour le shading
- Utilisé par Fragment et Vertex shaders
- Plusieurs choses utiles:
  - vec3, vec4, dvec4, mat4, sampler2D
  - mat*vec, mat*mat
  - reflect, refract
  - vec3 v(a.xy, 1)

VERTEX SHADER

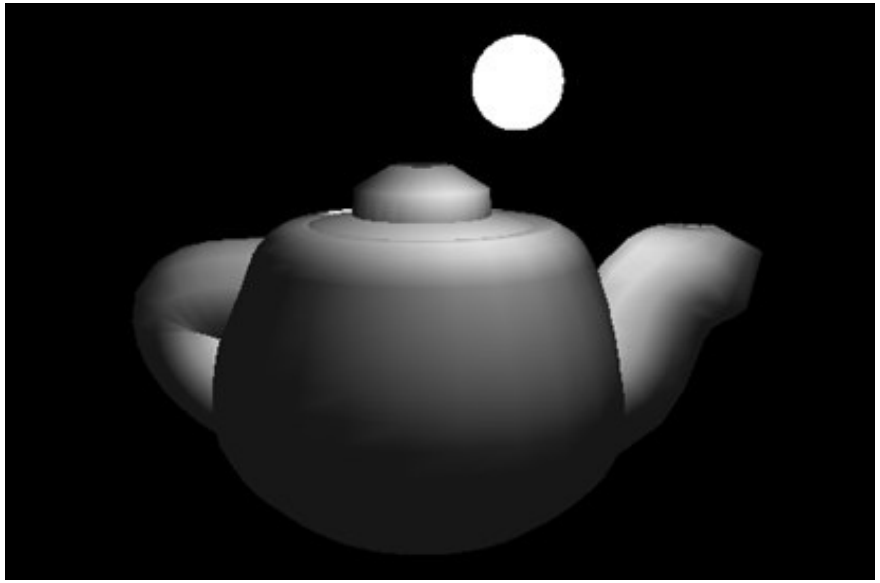
- VS fonctionne par chaque sommet séparément
- Par défaut, aucune notion de connectivité

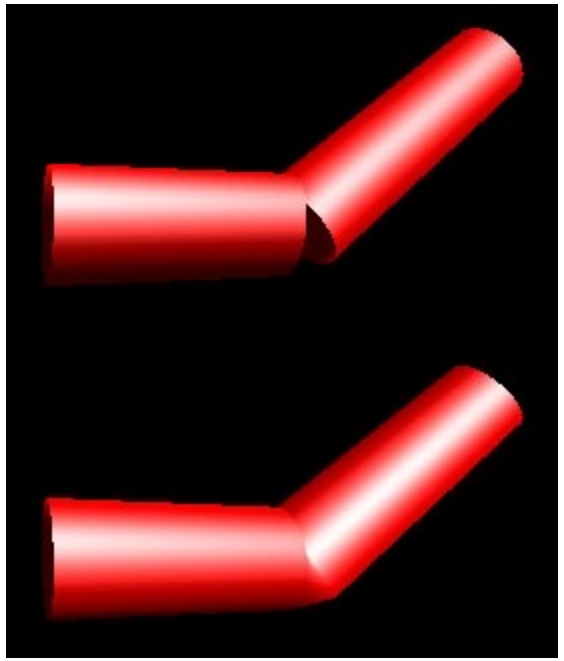
- Les données: les coordonnées dans le système de coordonnées de l'objet
- Son objectif principal est de définir **gl_Position**

Les coordonnées de l'objet → Les coordonnées du monde → Les coordonnées de VUE

VERTEX SHADER

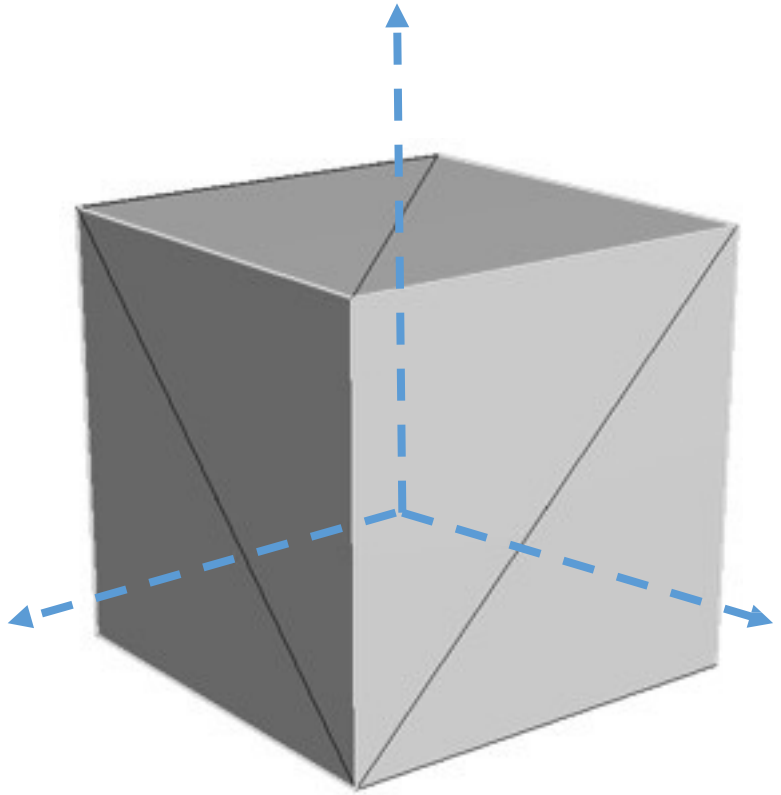
- En plus de convertir dans le système de coordonnées de vue
- On peut faire n'importe quoi avec les coordonnées (ou autres attributs)
  - e.g. déformer les sommets
  - e.g. skinning!



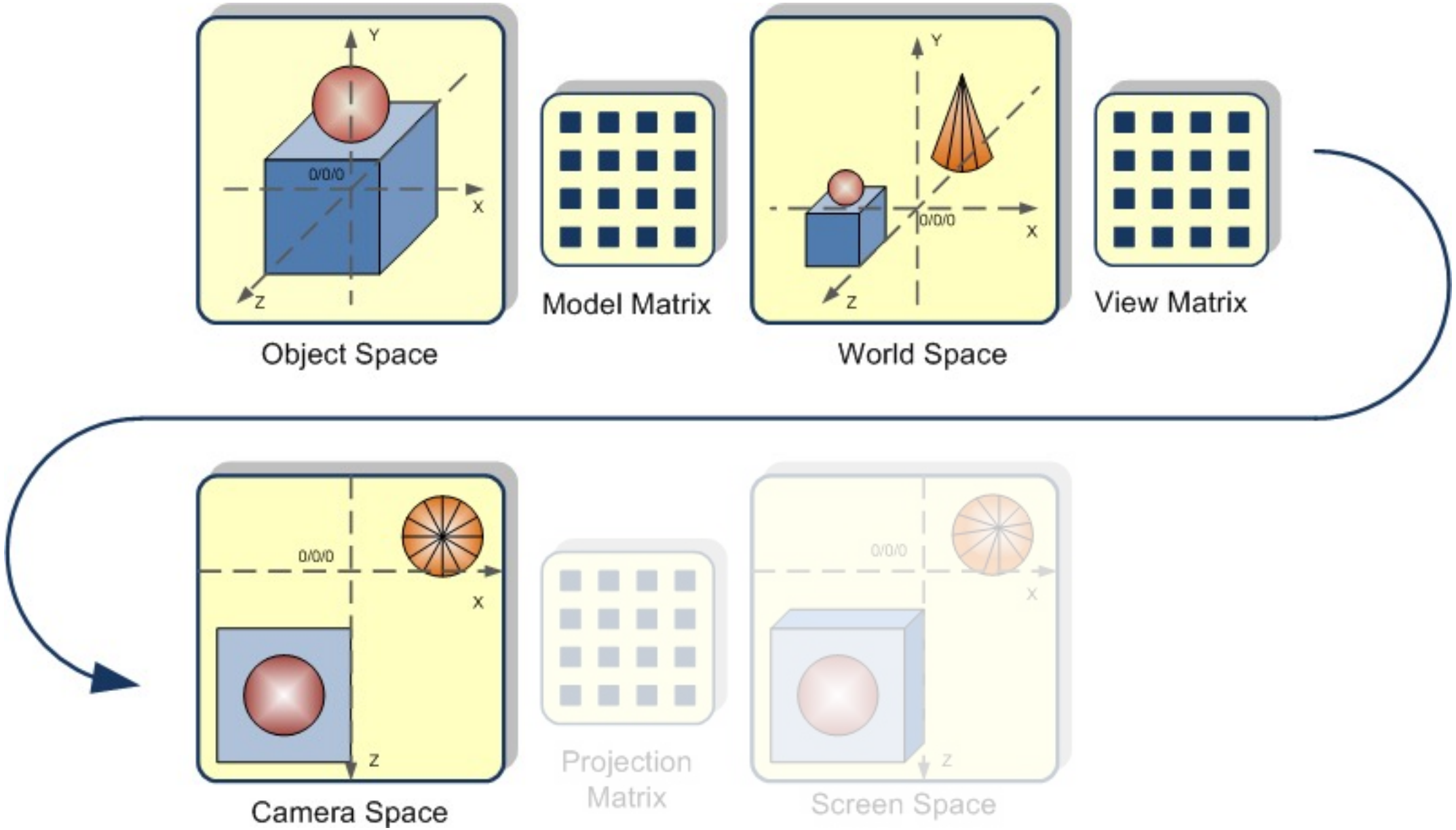


[courtesy NVIDIA]

```
var verticesOfCube = [
    -1,-1,-1,    1,-1,-1,    1, 1,-1,    -1, 1,-1,
    -1,-1, 1,    1,-1, 1,    1, 1, 1,    -1, 1, 1,
];
var indicesOfFaces = [
    2,1,0,    0,3,2,
    0,4,7,    7,3,0,
    0,1,5,    5,4,0,
    1,2,6,    6,5,1,
    2,3,7,    7,6,2,
    4,5,6,    6,7,4
];
var geometry = new THREE.PolyhedronGeometry(
verticesOfCube, indicesOfFaces, 6, 2 );
```



GÉOMETRIE (JAVASCRIPT/THRE.JS)



APERÇU DES TRANSFORMATIONS

- Toutes les transformations sont faites par des matrices 4x4:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix}$$

APERÇU DES TRANSFORMATIONS

- Toutes les transformations sont faites par des matrices 4x4:

Les coordonnées transformées

Les coordonnées du point

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix}$$

La matrice de transformation

APERÇU DES TRANSFORMATIONS

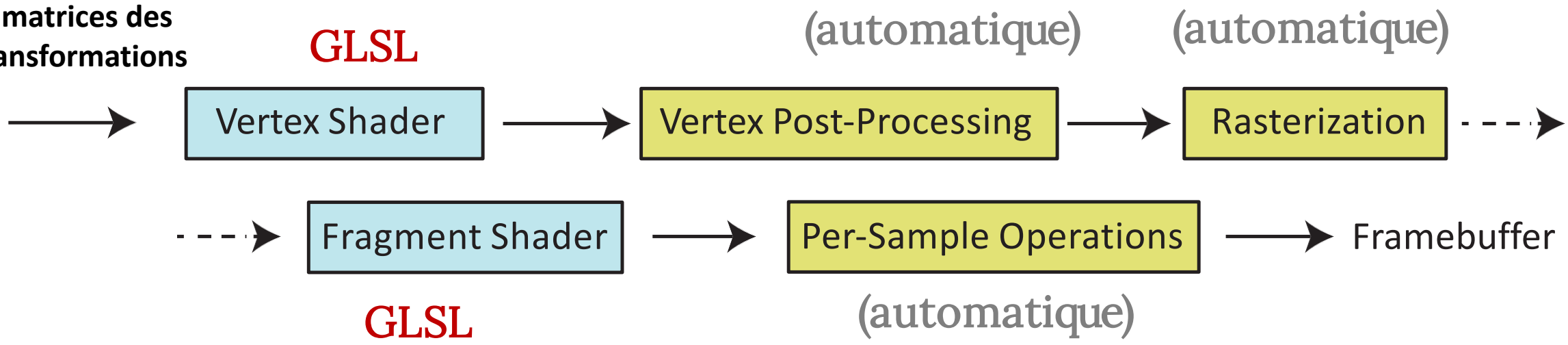
- Que fait cette transformation?

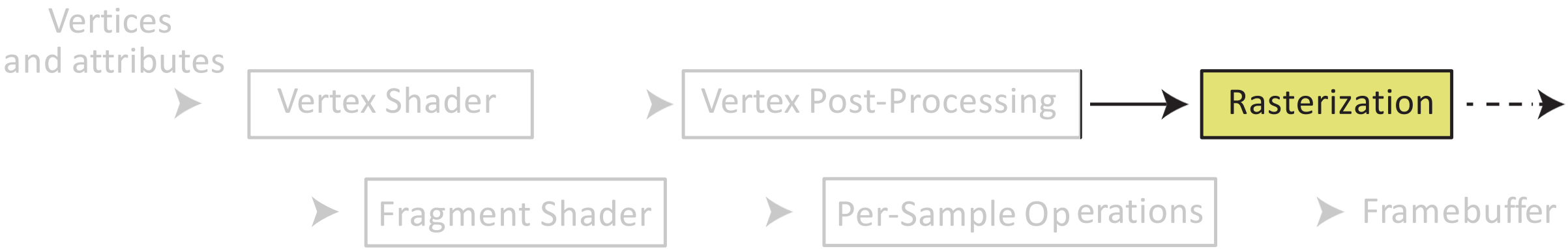
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} x \\ 2y \\ 3z \\ 1.0 \end{bmatrix}$$

OPENGL PIPELINE DE RENDU

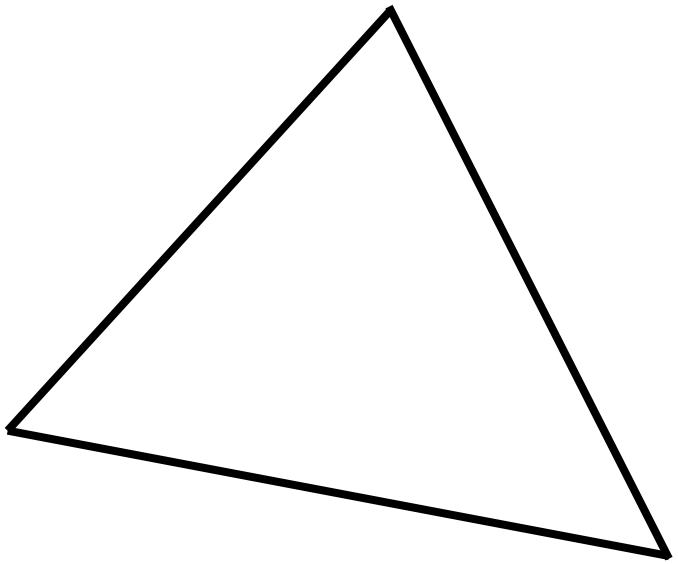
Javascript
+ Three.JS

Sommets
et attributs
+ matrices des transformations

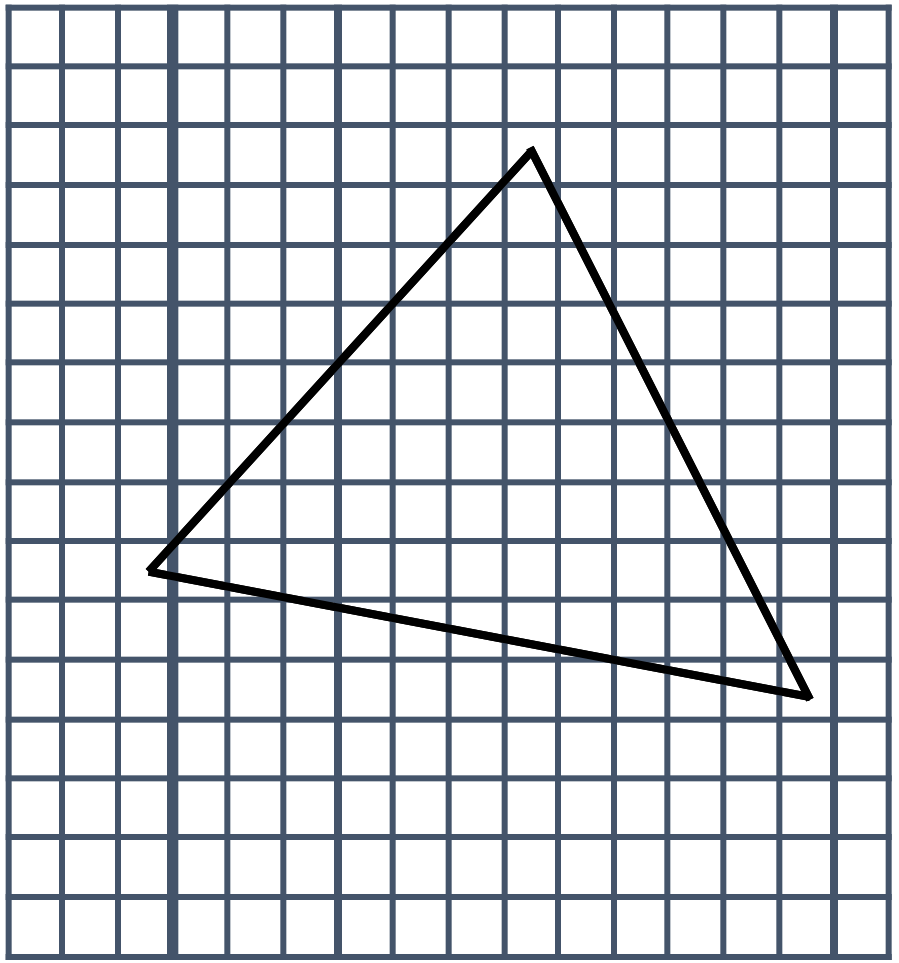



Vertices
and attributes
Vertex Shader
Vertex Post-Processing
Rasterization
Fragment Shader
Per-Sample Operations
Framebuffer

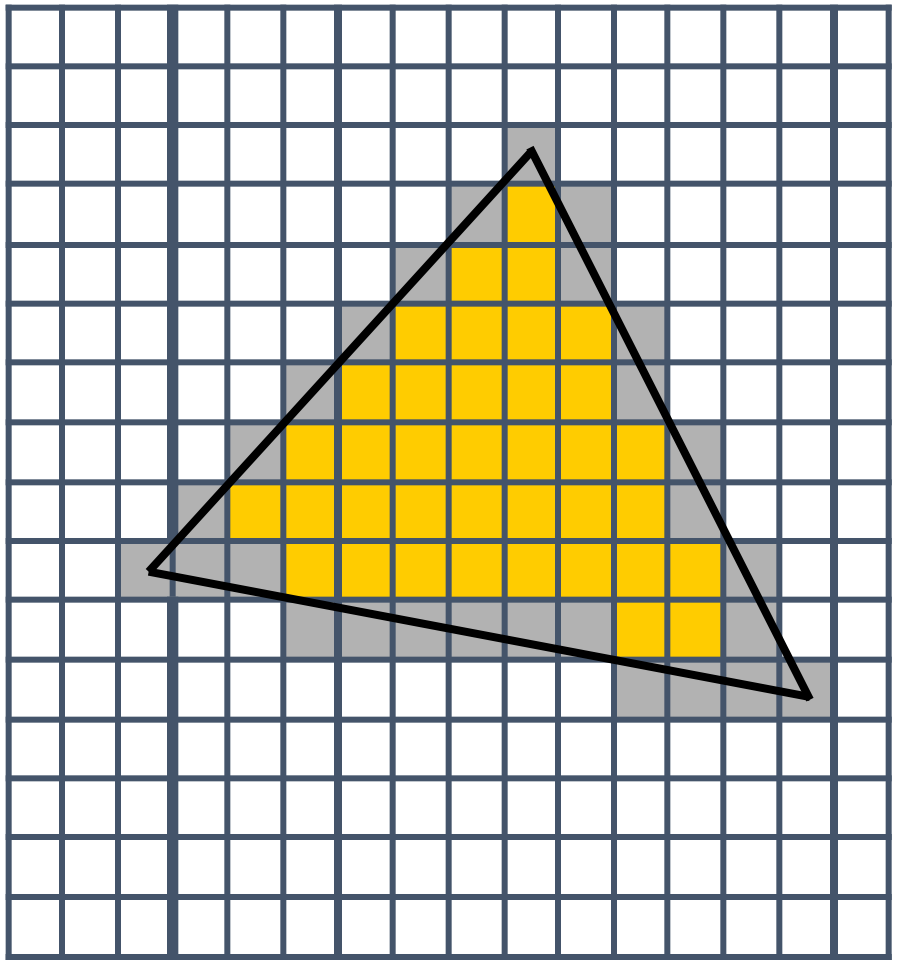
VUE DE LA CAMÉRA



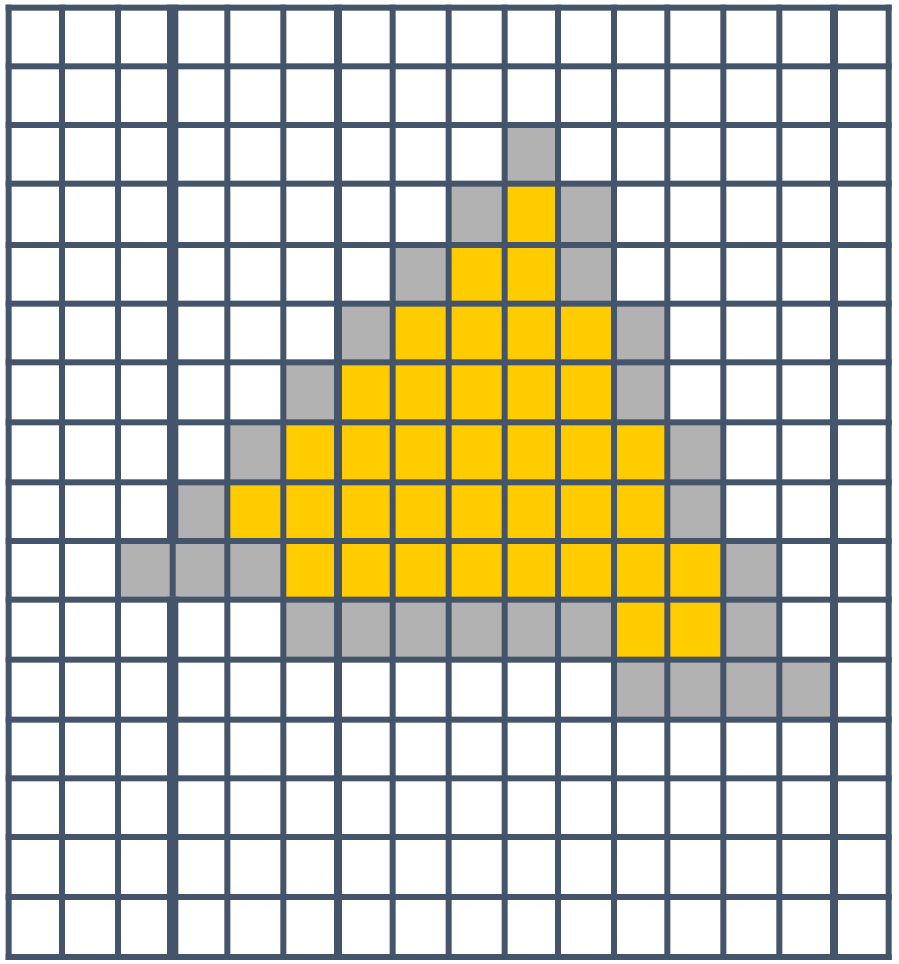
RASTERIZATION



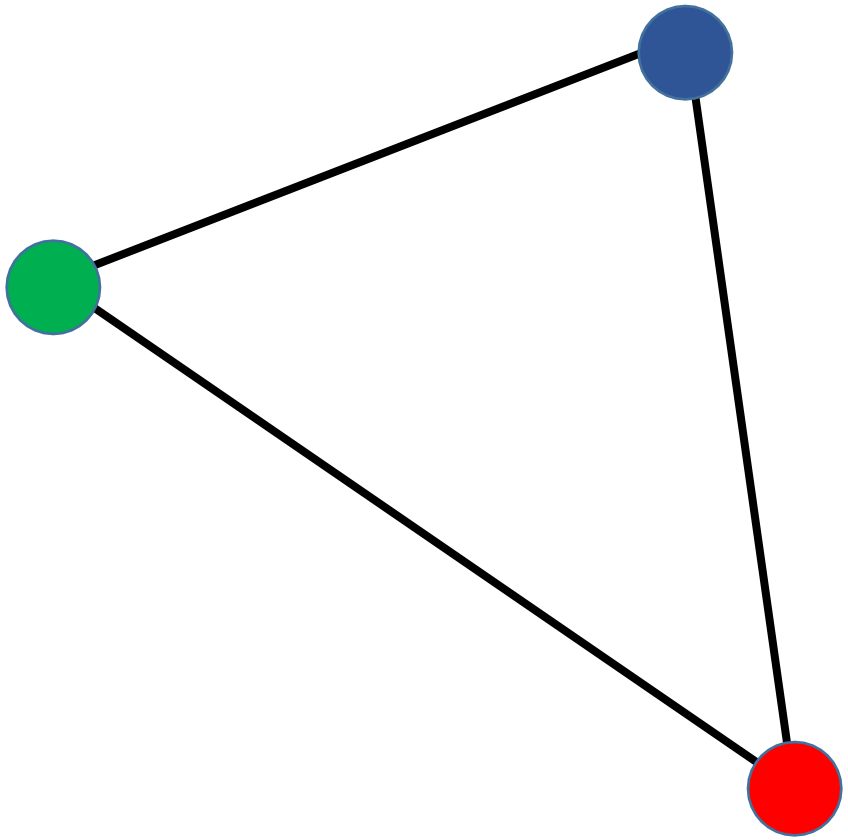
RASTERIZATION



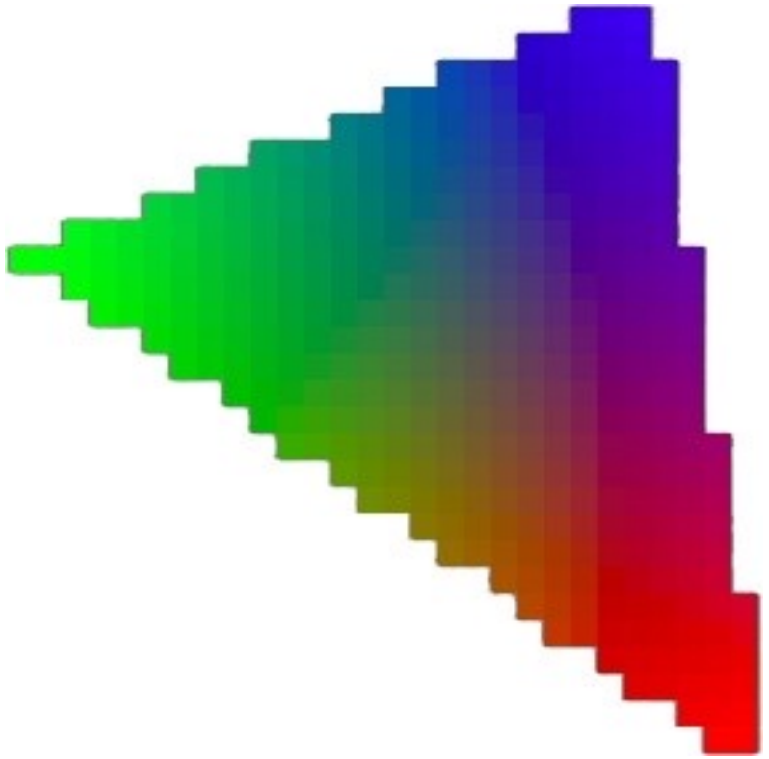
RASTERIZATION



RASTERIZATION - INTERPOLATION

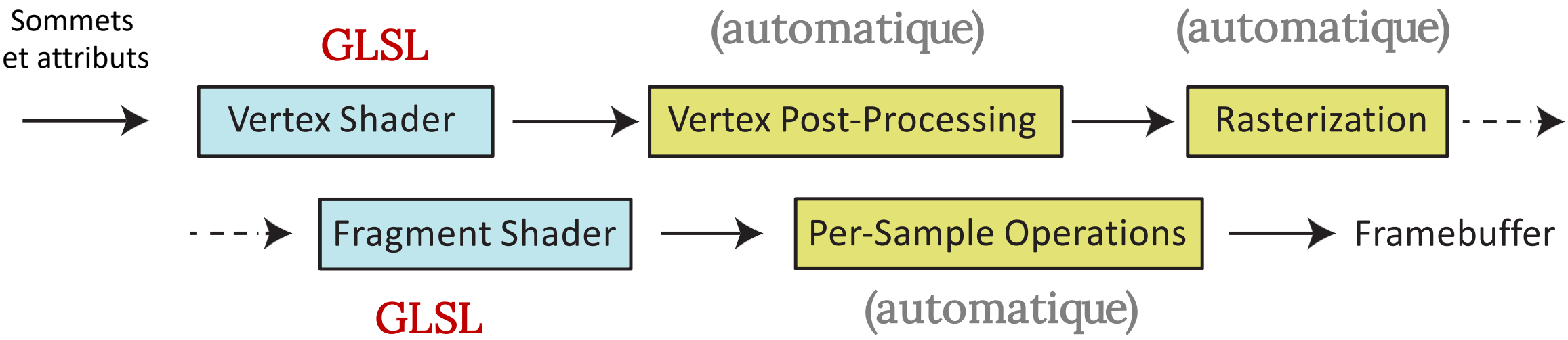


RASTERIZATION - INTERPOLATION



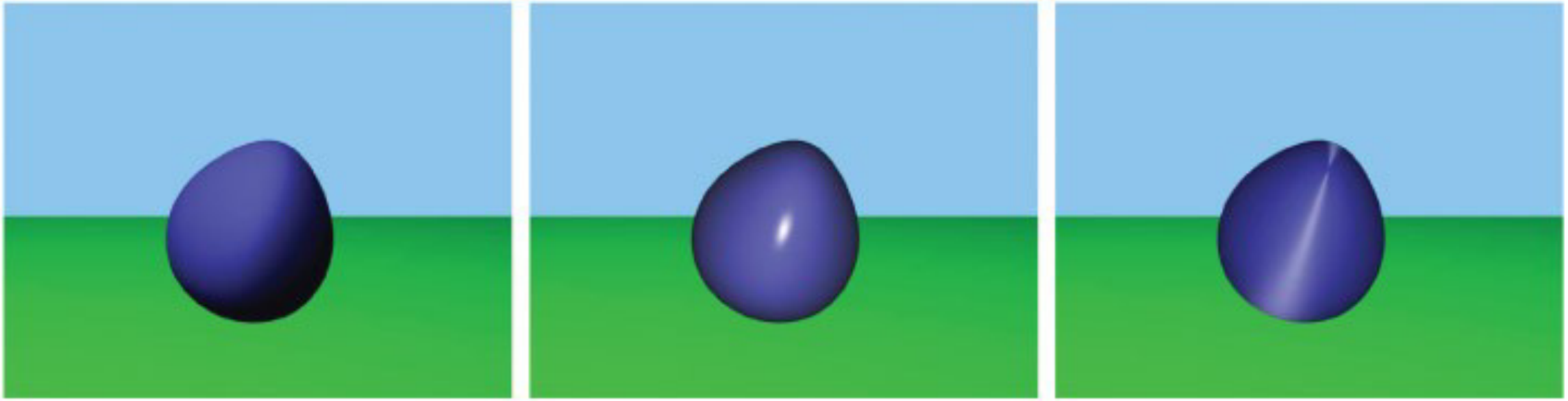
OPENGL PIPELINE DE RENDU

Javascript
+ Three.JS



FRAGMENT SHADER

- Fragment = les données pour affichage du pixel
- A gl_FragCoord – les coordonnées de l'affichage
- Peut définir la couleur!



FRAGMENT SHADER

- Tâches communes
  - texture mapping
  - Éclairage et shading par pixel

- Synonyme de Pixel Shader

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
}
```

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
}
```

Défini par Three.JS

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
}
```

Défini par Three.JS

$$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$$

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
```

$$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$$

Le système de coordonnées de la vue

Défini par Three.JS

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Rouge Vert Bleu Alpha

VERTEX SHADER – EXEMPLE 2

```
uniform float uVertexScale;
in vec3 vColor;
out vec3 fColor;



void main() {
 gl_Position = vec4(position.x * uVertexScale, position.y, 0.0,1.0);
 fColor = vColor;
}
```

VERTEX SHADER – EXEMPLE 2 (GLSL 1.0)

```
uniform float uVertexScale;
in attribute vec3 vColor;
out varying vec3 fColor;



void main() {
 gl_Position = vec4(position.x * uVertexScale, position.y, 0.0,1.0);
 fColor = vColor;
}
```

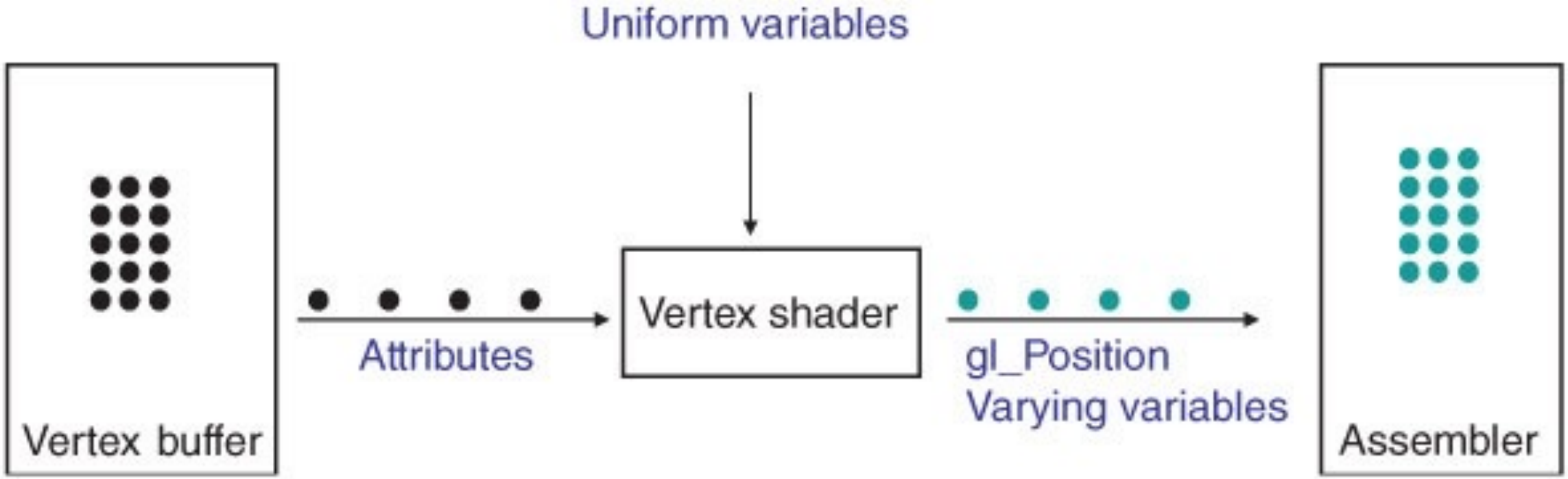
LES CONCEPTS

- uniform
  - La même chose pour tous les sommets ou fragments
- varying (**out** pour vertex shader, **in** pour fragment shader)
  - Calculé pour chaque sommet, puis interpolé automatiquement pour les fragments
- attribute (**in** pour vertex shader)
  - N'importe quelle valeur par sommet
  - Disponible uniquement dans Vertex Shader

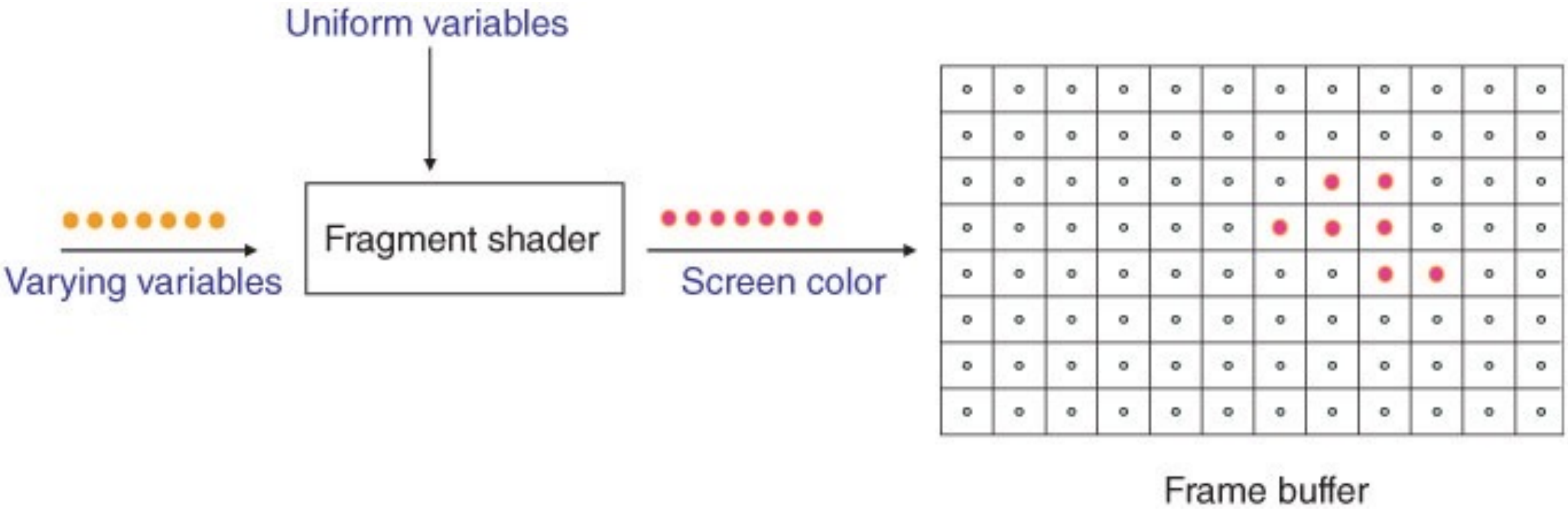
LES CONCEPTS

- uniform — JS + Three.JS → Vertex Shader → Fragment Shader
  - La même chose pour tous les sommets ou fragments
- varying — Vertex Shader → Fragment Shader
  - Calculé pour chaque sommet, puis interpolé automatiquement pour les fragments
- attribute — JS + Three.JS → Vertex Shader
  - N'importe quelle valeur par sommet
  - Disponible uniquement dans Vertex Shader

VERTEX SHADER



FRAGMENT SHADER



ATTACHING SHADERS

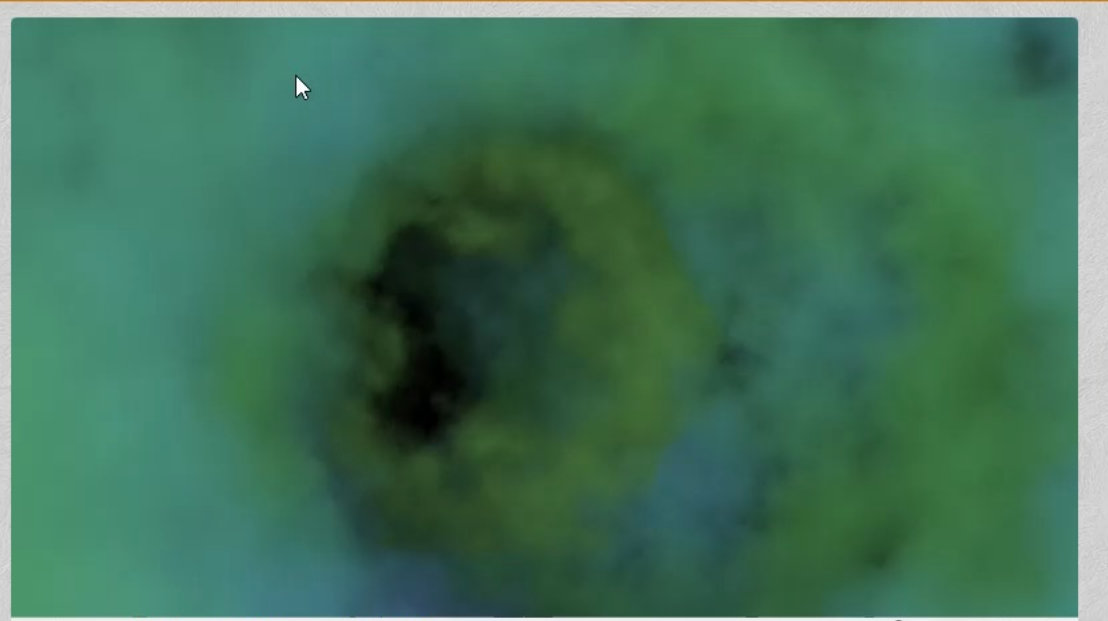
```
var remoteMaterial  = new THREE.ShaderMaterial({
  uniforms: {
  remotePosition:  remotePosition,
 },});
//voici le chargement des fichiers de shaders dans shaders[] …
remoteMaterial.vertexShader = shaders['glsl/remote.vs.glsl'];
remoteMaterial.fragmentShader = shaders['glsl/remote.fs.glsl'];
var remoteGeometry = new THREE.SphereGeometry(1, 32, 32);
var remote = new THREE.Mesh(remoteGeometry,  remoteMaterial);

scene.add(remote);
```

PLUS D’INFORMATION

https://threejs.org/docs/#api/en/renderers/webgl/WebGLProgram
https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf
http://www.shaderific.com/glsl/

SHADERTOY.COM

shadertoy.com/view/XljXRG



Cloudy stuff

54

Views: 7370, Tags: **noise**

Created by **Nrx** in 2015-10-12

Study of how to render cloudy stuff... I basically simplified Duke's **Cloudy spikeball** and merged it with the model I used for my **Love Tunnel**.

Comments (6)

Sign in to post a comment.

Jadwigga166, 2020-01-09
A very interesting idea. Makes an amazing impression.

okelly4408, 2019-12-29
Very nice, well done. Out of curiosity why do you make 'colorBackground' dependent on time and what not when simply setting it to black produces, at least I think so, an identical image?

dila, 2015-10-12
very cute. this one has been on my mind too

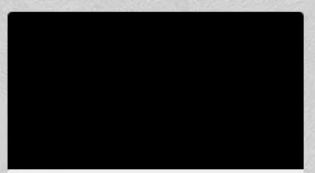
Image

Shader Inputs

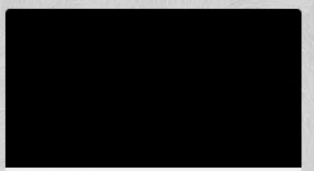
```glsl
// From https://www.shadertoy.com/view/XdBGDd
// From https://www.shadertoy.com/view/MljXDw (from Duke)

// Rendering parameters
#define CAMERA_FOCAL_LENGTH 3.0
#define RAY_STEP_MAX        100.0
#define RAY_LENGTH_MAX      150.0
#define NOISE_FACTOR        2.0
#define DIST_CORRECTION     0.6
#define DIST_MIN            0.6
#define DENSITY_FACTOR_STEP 0.01
#define DENSITY_FACTOR_DIST 0.3

// Math constants
#define PI  3.14159265359

// Rotation on the Z axis
vec3 vRotateZ (in vec3 p, in float angle) {
    float c = cos (angle);
    float s = sin (angle);
    return vec3 (c * p.x + s * p.y, c * p.y - s * p.x, p.z);
}

// Noise (from iq)
float noise (in vec3 p) {
    vec3 f = fract (p);
    p = floor (p);
    f = f * f * (3.0 - 2.0 * f);
    f.xy += p.xy + p.z * vec2 (37.0, 17.0);
    f.xy = texture (iChannel0, (f.xy + 0.5) / 256.0, -256.0).yx;
    return mix (f.x, f.y, f.z);
}

// FBM
float fbm (in vec3 p) {
    return noise (p) + noise (p * 2.0) / 2.0 + noise (p * 4.0) / 4.0;
}

// HSV to RGB
vec3 hsv2rgb (in vec3 hsv) {
    hsv.yz = clamp (hsv.yz, 0.0, 1.0);
```

Compiled in 0.0 secs — 2620 chars

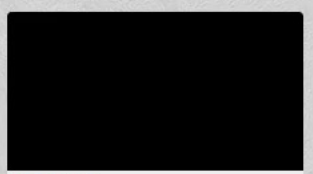
iChannel0



iChannel1



iChannel2



iChannel3

OPENGL RENDERING PIPELINE